Performance vs new features: it doesn’t have to be a zero-sum game

QCon London 2020

Dmitry Vyazelenko @DVyazelenko

Do you trust your file system?

Do you trust your file system?

All File Systems are Not Created Equal: On the Complexity of Crafting Crash Consistent Applications

Pillai et al. 2014

Do you trust your file system?

Persistence Property	ext2	ext2-sync	ext3-writeback	ext3-ordered	ext3-datajournal	ext4-writeback	ext4-ordered	ext4-nodelalloc	ext4-datajournal	btrfs	xfs	xfs-wsync	reiserfs-nolog	reiserfs-writeback	reiserfs-ordered	reiserfs-datajournal
Atomicity																
Single sector overwrite																
Single sector append	×		×			×								×		
Single block overwrite	×	×	×	×		×	×	×			×	×	×	×	×	
Single block append	×	×	×			×							×	×		
Multi-block append/writes	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
Multi-block prefix append	×	×	×			×							×	×		
Directory op	×	×											×			
Ordering																
Overwrite → Any op	×		×	×		×	×	×			×	×	×	×	×	
[Append, rename]→ Any op	×		×			×							×	×		
O_TRUNC Append → Any op	×		×			×							×	×		
Append → Append (same file)	×		×			×							×	×		
Append → Any op	×		×			×	×			×	×		×	×		
Dir op → Any op	×									×			×			

Table 1: Persistence Properties

Do you trust your file system?

Application	Silent errors	Data loss	Cannot open	Failed reads and writes	Other
Leveldb1.10	1	1	5	4	
Leveldb1.15	2		2	2	
LMDB					read-only open†
GDBM		2*	3*		
HSQLDB	2	3	5		
Sqlite-Roll		1*			
Sqlite-WAL					
PostgreSQL			1†		
Git		1*	3*	5*	3#*
Mercurial		2*	1*	6*	5 dirstate fail*
VMWare			1*		
HDFS			2*		
ZooKeeper		2*	2*		
Total	**5**	**12**	**25**	**17**	**9**

Table 2: Failure Consequences

Do you trust your file system?

[Redundancy does not imply fault tolerance: analysis of distributed storage reactions to single errors and corruptions]

Ganesan et al., *FAST 2017*

Do you trust your file system?

We studied eight widely used distributed storage systems: Redis, ZooKeeper, Cassandra, Kafka, RethinkDB, MongoDB, LogCabin, and CockroachDB.

We find that these systems can silently return corrupted data to users, lose data, propagate corrupted data to intact replicas become unavailable, or return an unexpected error on queries.

“Don't use ZFS. It's that simple. It was always more of a buzzword than anything else, I feel, and the licensing issues just make it a non-starter for me.”

– Linus Torvalds

https://www.realworldtech.com/forum/?threadid=189711&curpostid=189841

CRC

A **cyclic redundancy check** (**CRC**) is an error-detecting code commonly used in digital networks and storage devices to detect accidental changes to raw data. Blocks of data entering these systems get a short *check value* attached, based on the remainder of a polynomial division of their contents.

CRC in Java 8

CRC in Java 8

```java
package java.util.zip;

public interface Checksum
{
  public void update(int b);

  public void update(byte[] b, int off, int len);

  public long getValue();

  public void reset();
}
```

CRC in Java 8

```java
public class CRC32 implements Checksum
{
  /**
   * …
   * Upon return, the buffer's position will
   * be updated to its limit; its limit will not have been
   * changed.
   *
   * @param buffer the ByteBuffer to update the checksum with
   * @since 1.8
   */
  public void update(ByteBuffer buffer)
}
```

CRC in Java 14

```java
public interface Checksum
{
  public void update(int b);

  default public void update(byte[] b) // @since 9

  public void update(byte[] b, int off, int len);

  default public void update(ByteBuffer buffer) // @since 9

  public long getValue();

  public void reset();
}
```

CRC in Java 8

CRC in Java 8

```
final ByteBuffer buffer = ...;
final int position = buffer.position();

final CRC32 checksum = new CRC32();
checksum.update(buffer);
final int checksum = (int)checksum.getValue();

buffer.position(position);
```

CRC in Java 8

```
final ByteBuffer buffer = ...;
final int position = buffer.position();

final CRC32 checksum = new CRC32();
checksum.update(buffer);
final int checksum = (int)checksum.getValue();

buffer.position(position);
```

How about...

```java
public static int crc32(int crc, ByteBuffer buffer,
                        int offset, int length)
```

How about…

```java
public static int crc32(int crc, ByteBuffer buffer,
                        int offset, int length)
```

```java
public static int crc32(int crc, long address,
                        int offset, int length)
```

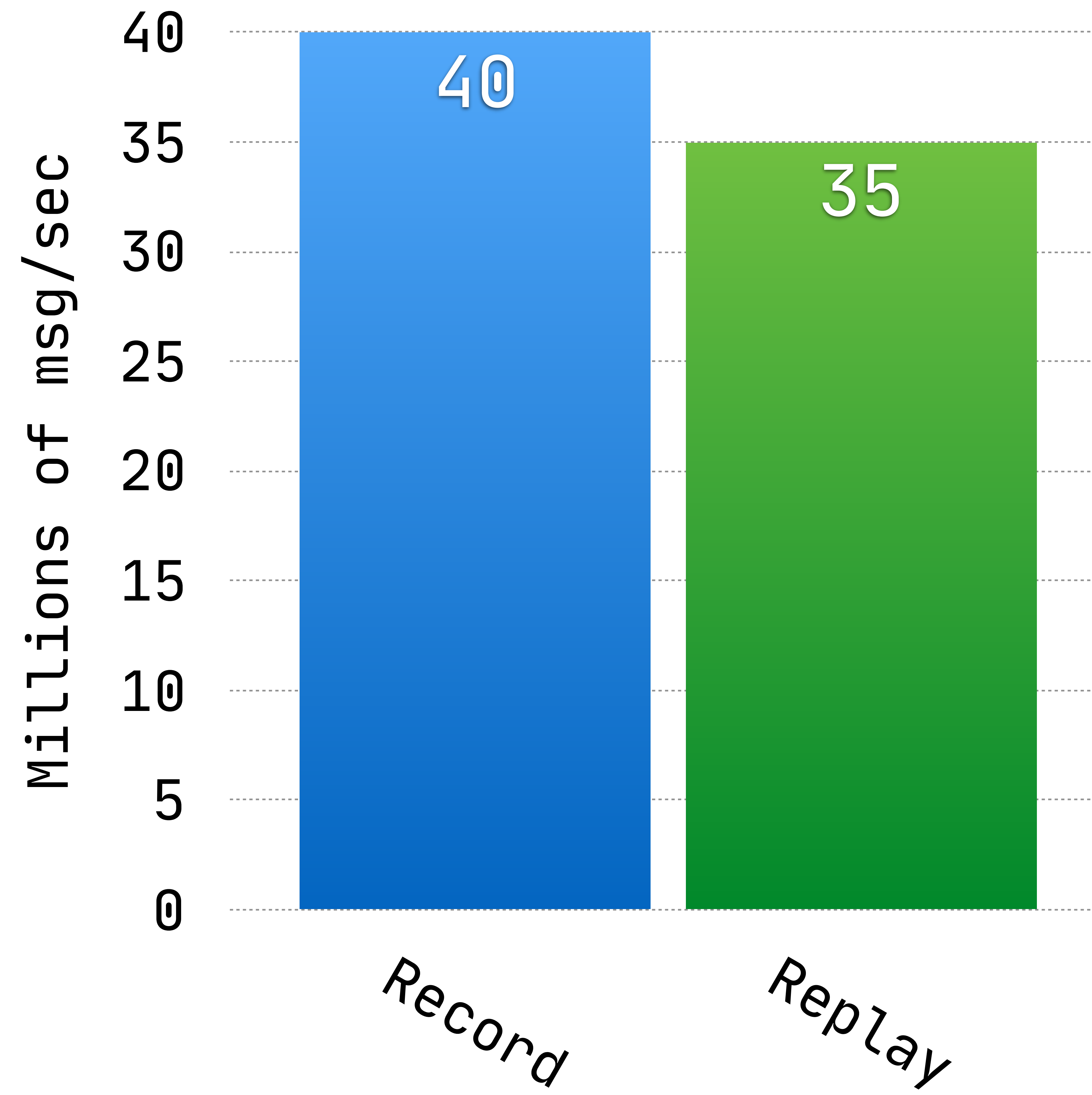
Digging deeper

```
public class CRC32 implements Checksum
{

  private native static int update(int crc, int b);

  private native static int updateBytes(int crc, byte[] b,
                                        int off, int len);

  private native static int updateByteBuffer(int adler,
                                             long addr,
                                             int off,
                                             int len);
}
```

Digging deeper

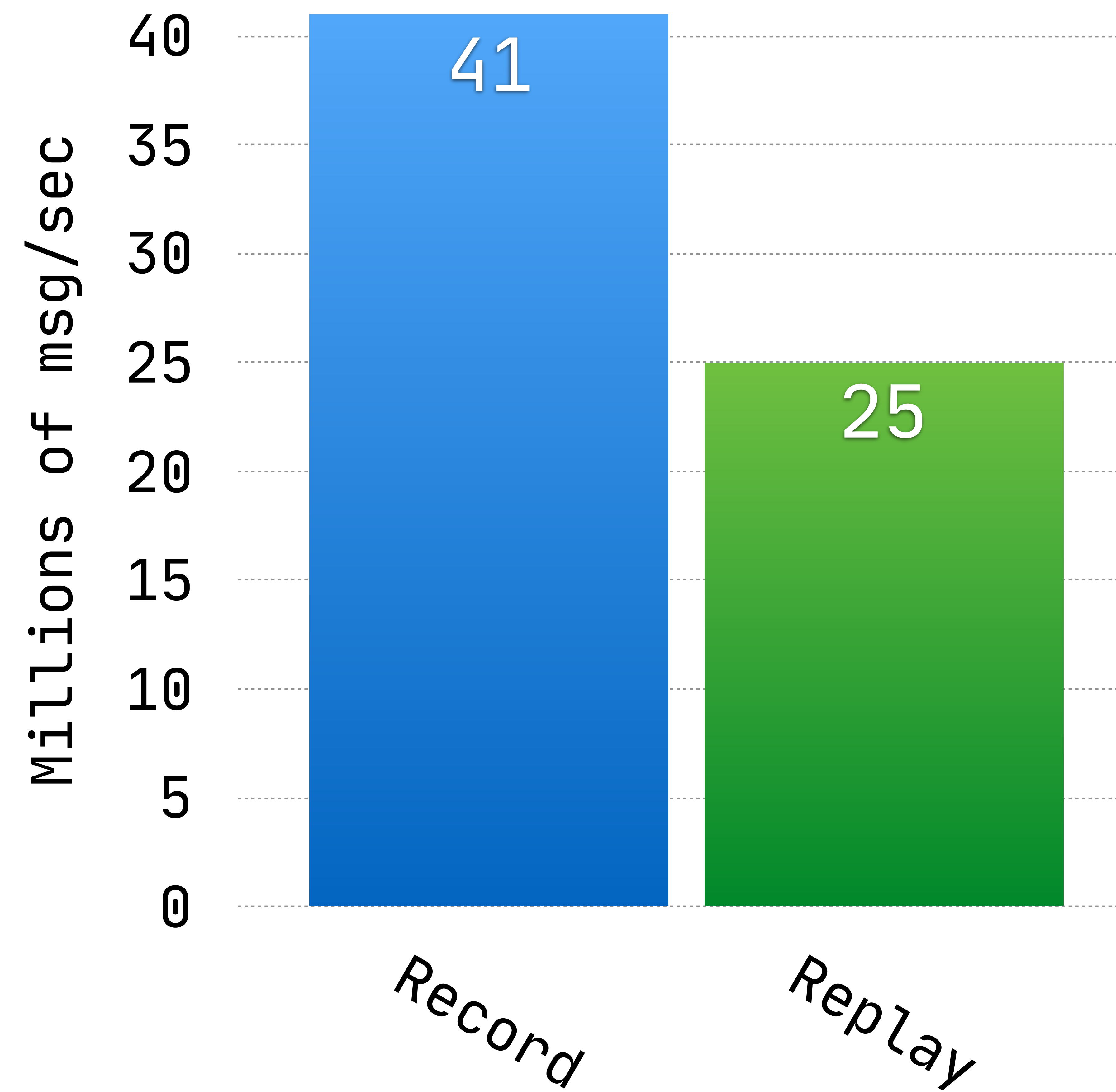
```
public final class CRC32C implements Checksum
{

  @HotSpotIntrinsicCandidate
  private static int updateBytes(int crc, byte[] b,
                                 int off, int end)

  @HotSpotIntrinsicCandidate
  private static int updateDirectByteBuffer(int crc,
                                            long address,
                                            int off,
                                            int end)
}
```

Baseline

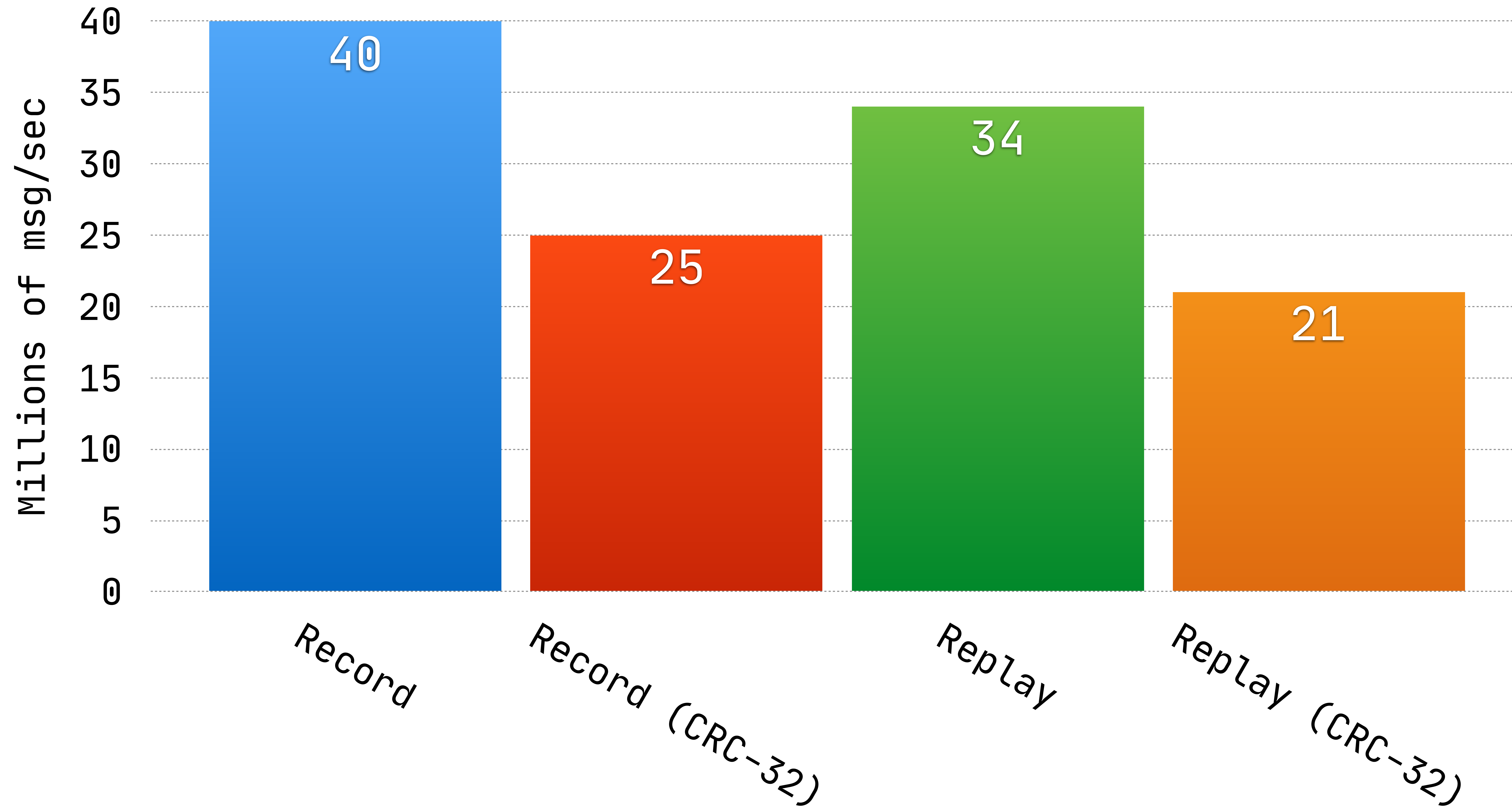


Baseline

JDK 11 (11.0.6+10-LTS)

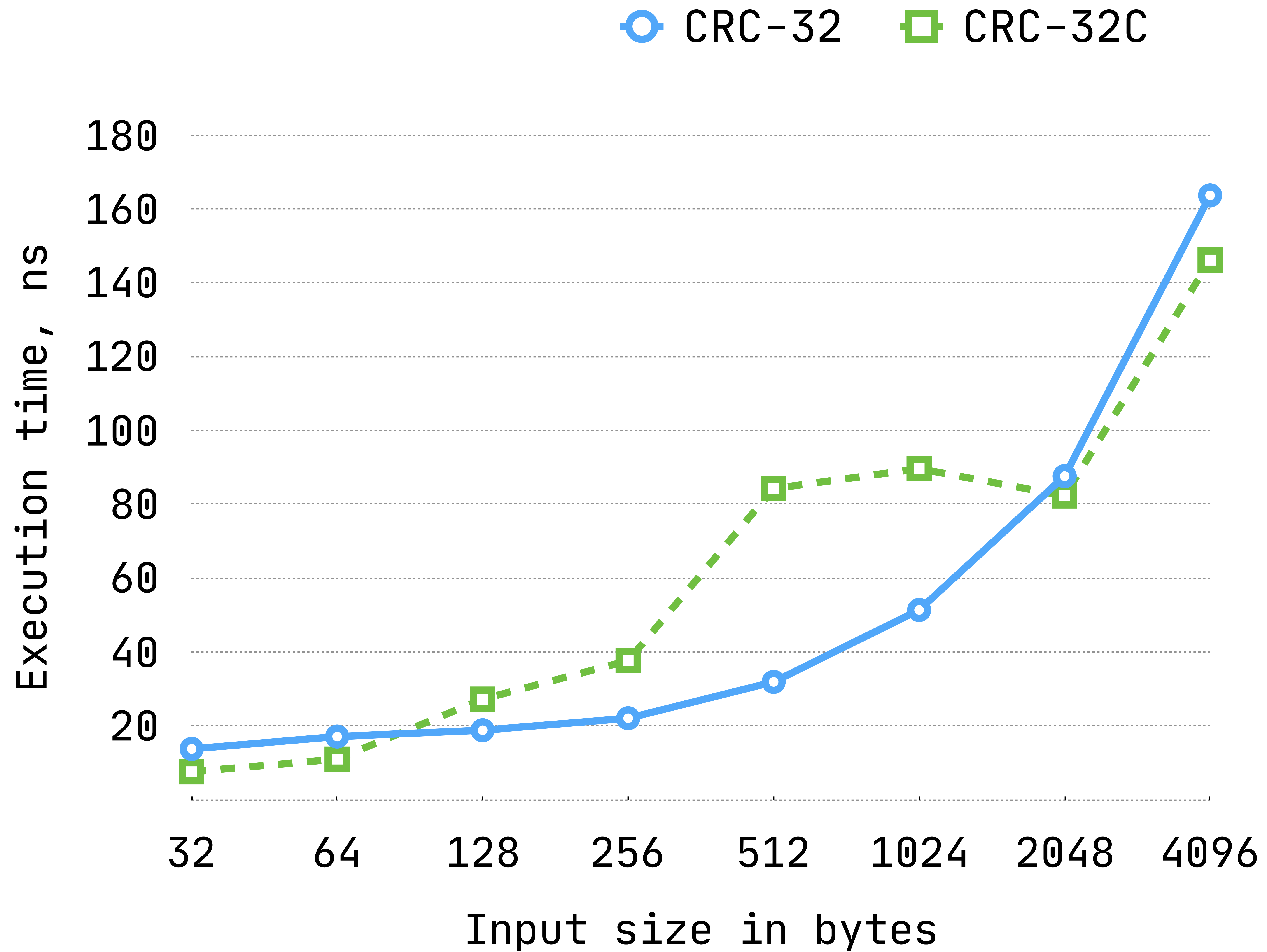


Initial CRC support

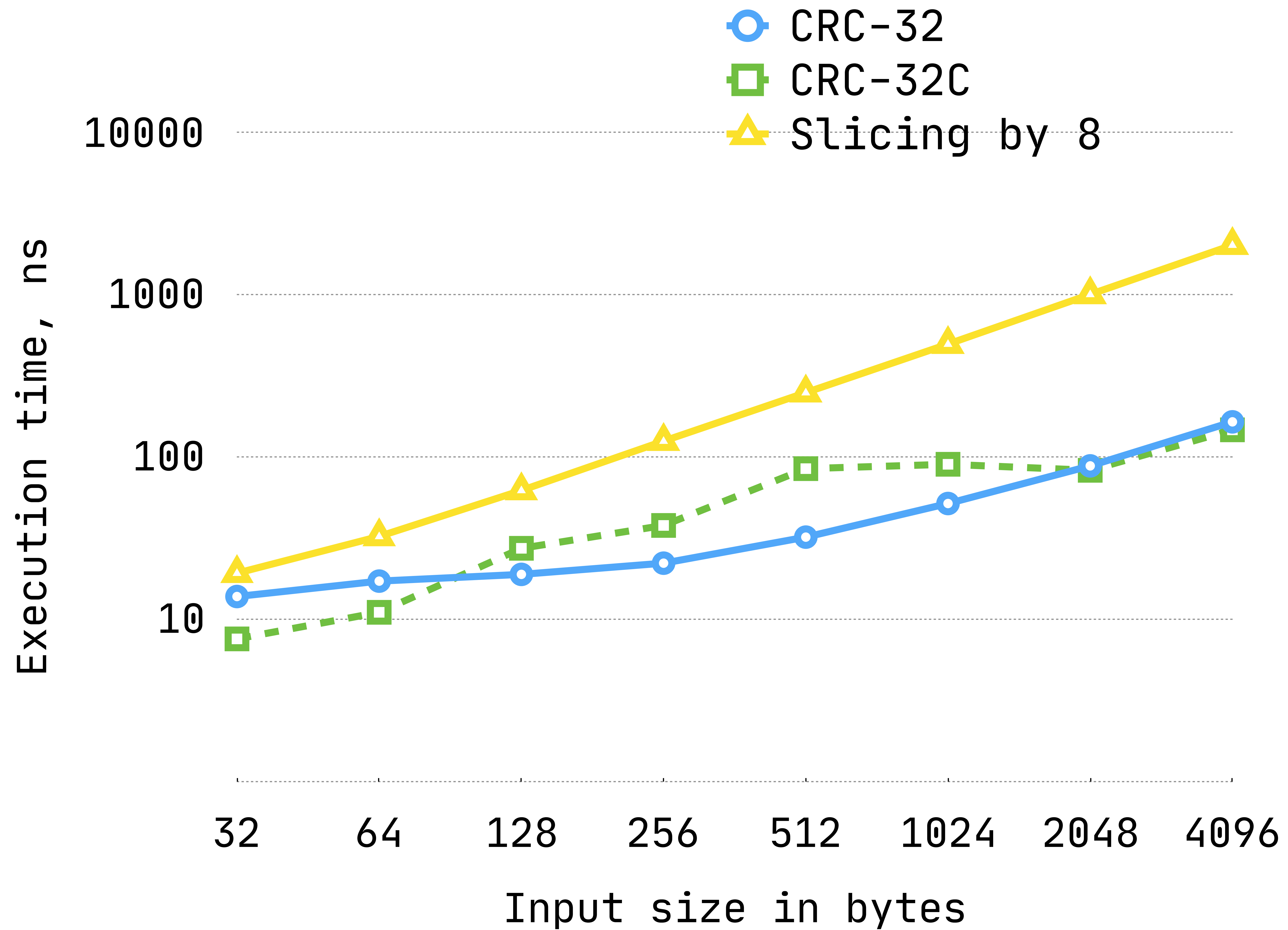


Initial CRC support

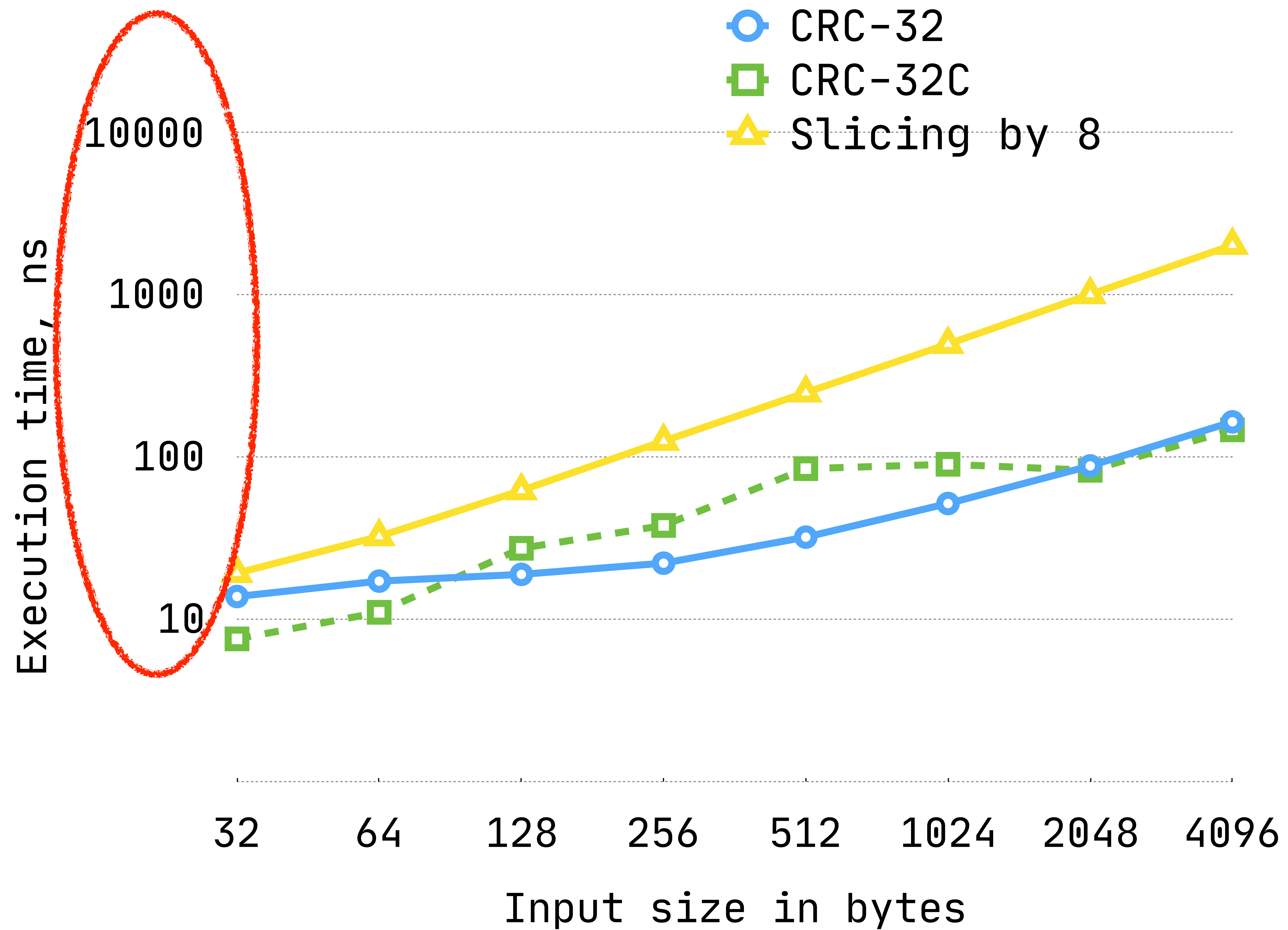
CRC-32 vs CRC-32C



CRC-32 vs CRC-32C



CRC-32 vs CRC-32C



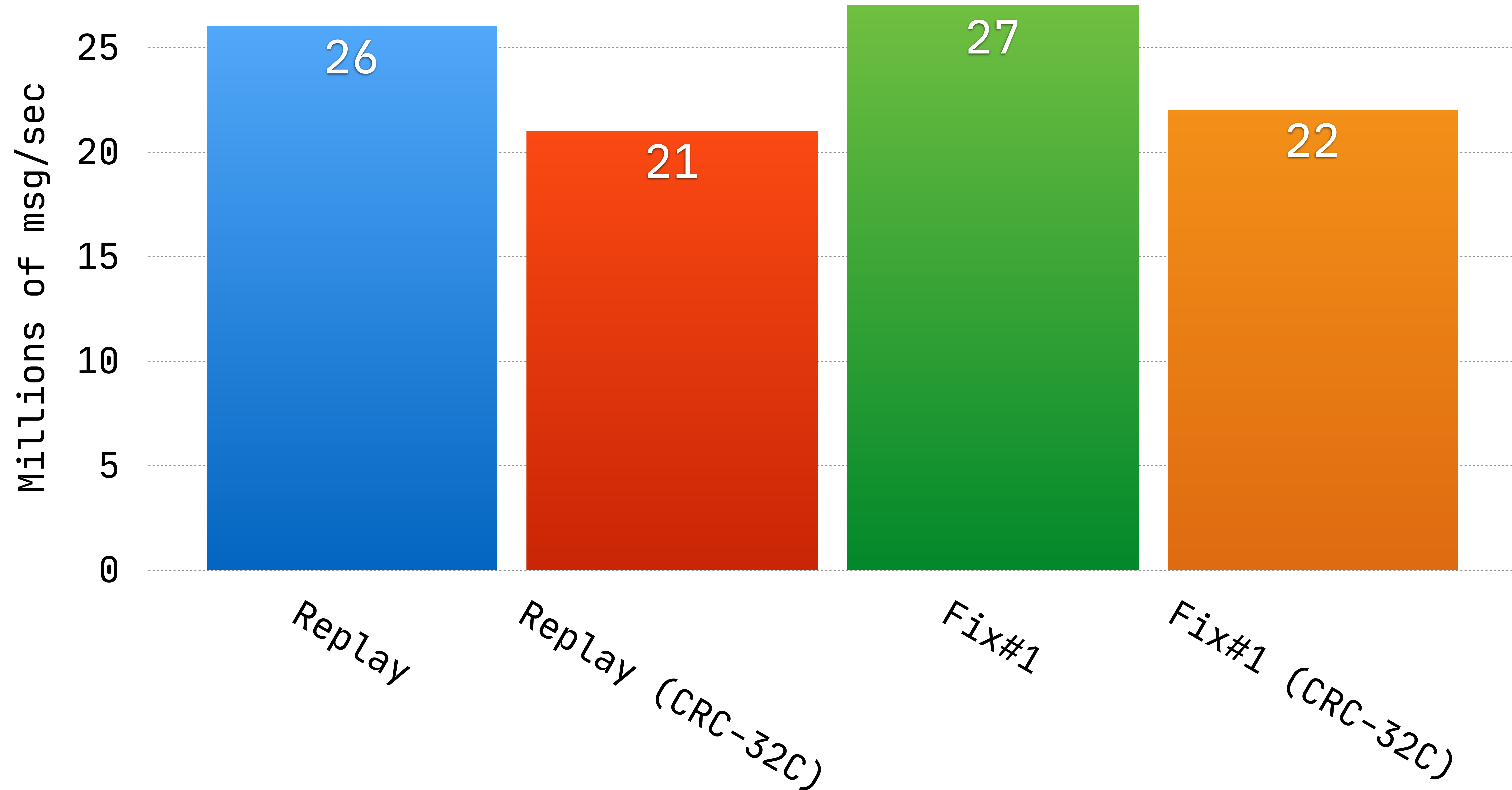
How replay works

```java
final int bytesRead = readRecording();

while (hasMoreFrames(bytesRead))
{
    final int frameLength = readFrame();
    verifyChecksum();
    if (publication.tryClaim(frameLength, bufferClaim))
    {
        bufferClaim.putBytes(replayBuffer, offset, frameLength)
                   .commit();
    }
}
```

Let's fix replay #1

```
final int bytesRead = readRecording();

while (hasMoreFrames(bytesRead))
{
    final int frameLength = readFrame();
    verifyChecksum();
    handleStartOfBatch();
    if (publication.tryClaim(frameLength, endClaim))
    {
        endClaim.putBytes(replayBuffer, offset, frameLength)
                   .commit();
    }
    handleEndOfBatch();
}
```

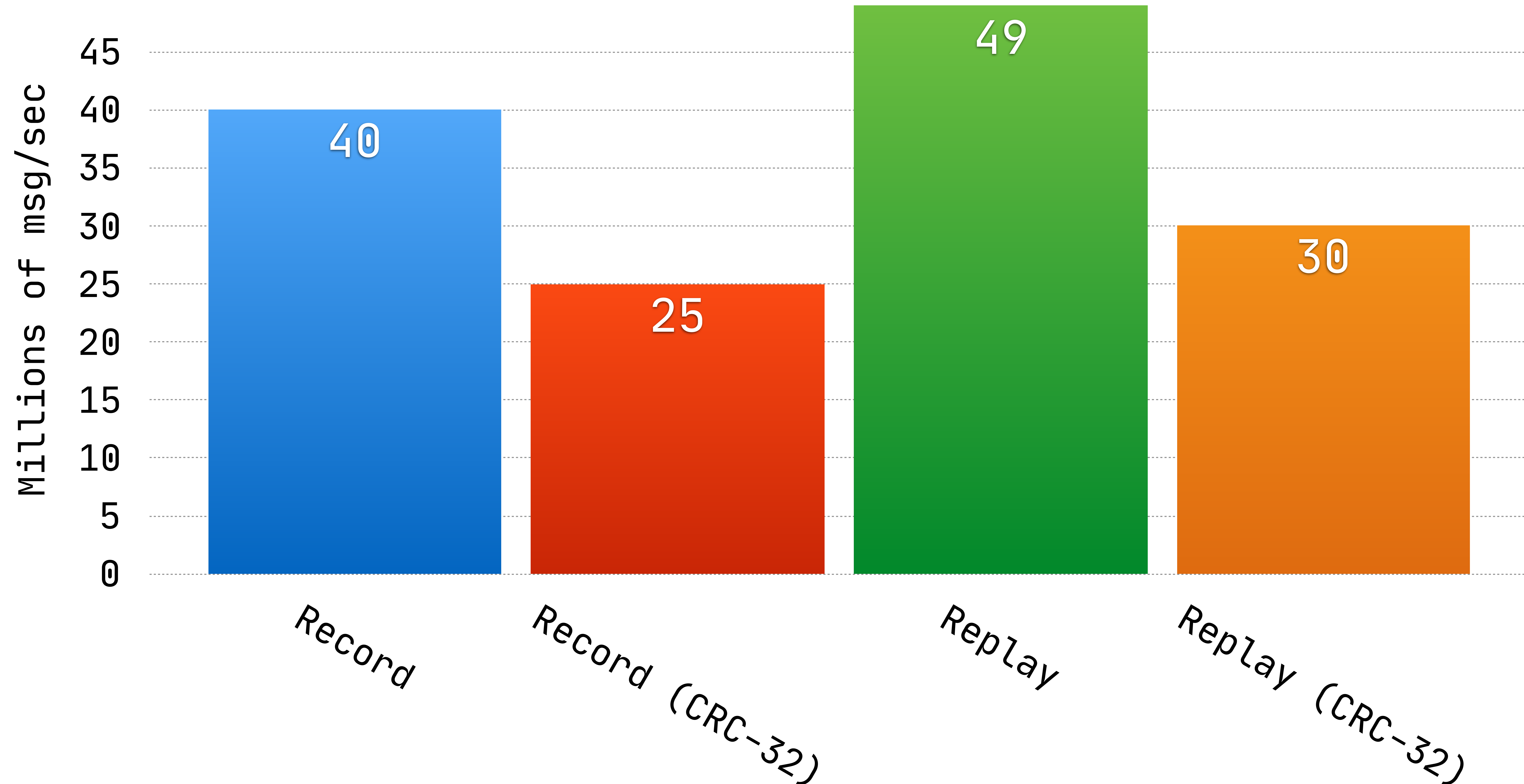
Let's fix replay #1



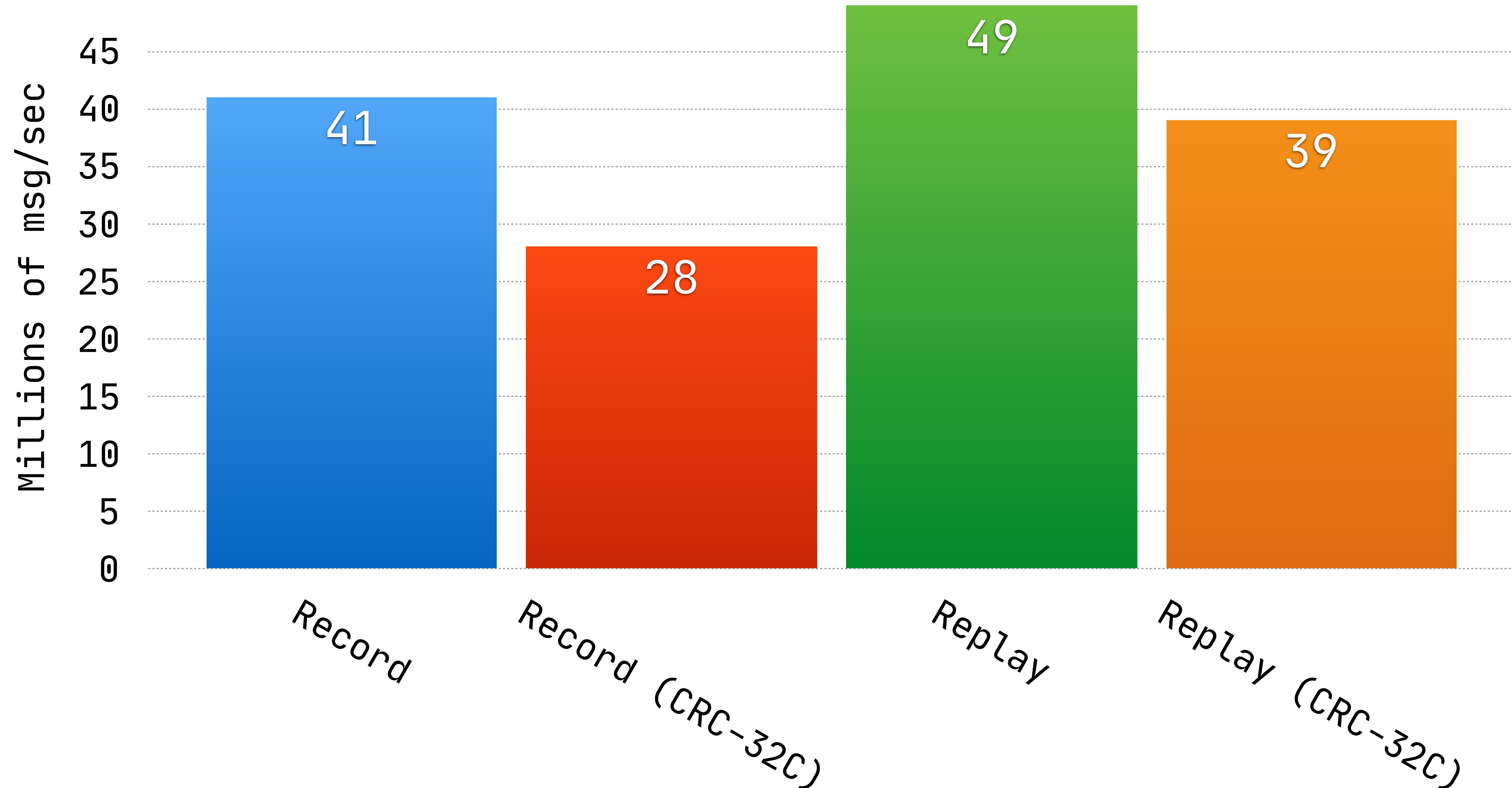
Let's fix replay #2

```
final int bytesRead = readRecording();

while (hasMoreFrames(bytesRead))
{
    readFrame();
    verifyChecksum();
    prepareFrame();
}

publication.offerBlock(replayBuffer, 0, endOfLastFrame);
```

Let's fix replay #2



Let's fix replay #2



Conclusions

- Performance vs new features: it doesn't have to be a zero-sum game
- Stay curious and keep on digging…